An Event-Centric Approach to Authoring Stories in Crowds

Mubbasir Kapadia[1], Alexander Shoulson[2], Cyril Steimer[2], Samuel Oberholzer[2], Robert W. Sumner[2,3], Markus Gross[2,3]
[1]Rutgers University, [2]Disney Research Zurich [3]ETH Zurich

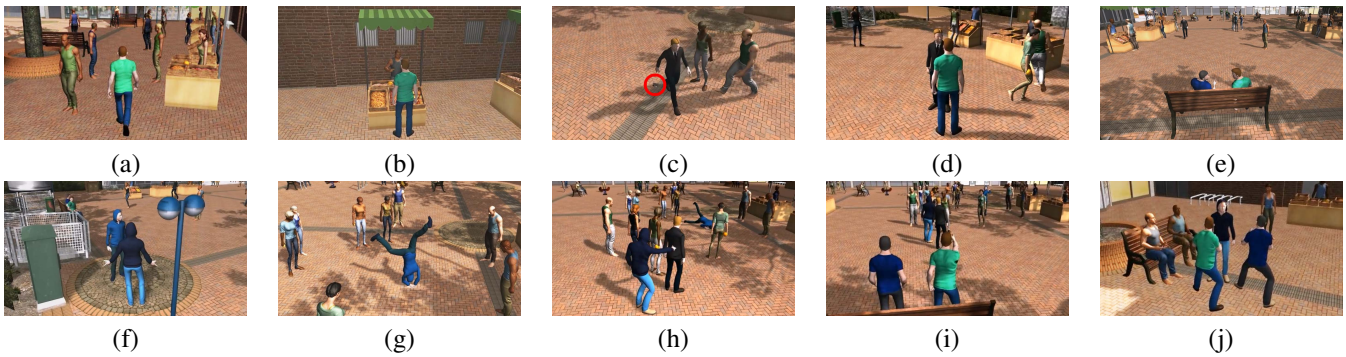

Figure 1: *A complex narrative authored using our framework. (a) Our protganist, Tom walks into a busy marketplace. (b) He haggles with some vendors to buy some fruit. (c) Meanwhile, a businessman loses his wallet. (d) Fortunately, Tom retrieves the wallet and returns it to him. (e) Tom and his friend engage in a conversation and enjoy the view while seated on a bench. (f) Elsewhere, two suspicious men scheme to steal from the businessman while Tom looks on. (g) One of the men distracts the crowd, (h) while the other steals the wallet from the distracted businessman. (i) Fortunately Tom spots the thief, and (j) catches him before he gets away.*

Abstract

We present a graphical authoring tool for creating complex narratives in large, populated areas with crowds of virtual humans. With an intuitive drag-and-drop interface, our system enables an untrained author to assemble story arcs in terms of narrative events that seamlessly control either principal characters or choreographed heterogeneous crowds within the same conceptual structure. Smart Crowds allow groups of characters to be dynamically assembled and scheduled with ambient activities, while also permitting individual characters to be selected from the crowd and featured more prominently as an individual in a story with more sophisticated behavior. Our system runs in real-time at interactive rates with no pause or costly pre-computation step between creating a story and simulating it, making this approach ideal for storyboarding or previsualization of narrative sequences.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: crowd authoring, crowd animation

1 Introduction

The process of simulating vibrant, living virtual environments is dependent on producing and managing the behavior of autonomous virtual characters in fully-realized 3D worlds. Stories told and shown in busy spaces, such as those of a populous city or busy shopping area, are dependent on an active backdrop of human characters performing a level of ambient activity to give life to the environment. These characters must exhibit intelligence not only in their individual routine, but in their interactions with one another and with the environment itself in order to present a realistic representation of expected human behavior.

We are interested in creating narratives in highly authorable environments full of active and involved virtual humans. Just as in life, where stories worth telling can occur at any time and involve any number of actors, our goal is to give story authors complete flexibility when conducting the behavior of a virtual populace – freely deciding where a narrative takes place and who it involves. At the same time, however, we are limited in computational capacity and cannot afford to fully simulate every character always at the highest fidelity in real-time. Even regardless of computational cost, the sheer complexity of individual behavior authoring grows intractably from a *conceptual* standpoint; as the number of virtual characters increases beyond even just a few dozen, those actors become more difficult to design, diversify, and debug.

Numerous approaches exist for creating stories and controlling crowds of actors, but few systems combine the two techniques into a single solution. Individual behavior authoring or centralized drama managers can be an effective tool for building stories from the decision processes of virtual actors, but such systems are traditionally limited to conducting a small number of richly-simulated characters at a time. Conversely, crowd simulation techniques allow the production and management of potentially thousands of virtual humans, but the characters in these systems typically do not survive close individual scrutiny when trying to preserve a user's sense of realism and engagement. In stories, large crowds can be instructed to respond to stimuli by broadcasting homogeneous commands (like making every individual gaze at a nearby explosion), but these actions lack diversity and nuance and provide no individualized control to an author. More often, if crowds appear at all, they serve as decoration in the backdrop of a story focusing on an immutable set of principal, high-fidelity main characters.

MiG '16,, October 10-12, 2016, Burlingame, CA, USA
ISBN: 978-1-4503-4592-7/16/10
DOI: http://dx.doi.org/10.1145/2994258.2994265

Many technical and communication challenges exist in trying to break the divide between crowds and principal characters for use in authored stories. Our work examines three primary questions:

- **How do we represent complex worlds with many actors and objects using accessible metaphors to an untrained author?** That is, how do we mitigate the complexity of controlling over 100 characters and objects in a virtual world and present the user with a manageable set of meaningful and interesting choices to make when crafting a story? More broadly, how do we decouple authoring complexity from simulation scale?
- **How do we promote actors from being part of a crowd to principal characters that drive the story forward, and vice-versa?** Since our stories could feasibly involve any character, all of our actors must be able to exhibit high-quality behaviors on demand. However, for both computation and conceptual efficiency, it becomes necessary to create grouped crowds that act according to more centralized decision-making.
- **How do we make crowds themselves an accessible and useful part of the stories our authors want to create?** Being that our virtual worlds consist of large numbers of virtual humans and objects, how do we encapsulate our crowds to make them available to an author for meaningful inclusion in a story? How do we design our crowd system so as to be more than an animated backdrop to the stories we mean to tell?

Our work tackles not only computational challenges, but the problem of exposing the power of story authoring and virtual character intelligence at scale to untrained authors.

Where typical crowd or narrative simulations focus either on large homogeneous crowds or small numbers (2-15) of expensively simulated virtual characters, our system achieves both breadth and depth on a flexible basis. We do this using a hierarchy of distributed and centralized behavior controllers that provide both computational simplicity and accessible metaphors for explaining the world and the activity of its populous to an untrained author. Our approach has two main aspects.

First, we provide an authoring tool that allows a storyteller to design plots based on atoms of narratively significant character interactions. We use event-centric behavior, a semi-centralized system where interactions ("events") temporarily co-opt the behavior of their participants, guide them through an interaction, and then release them when finished. Events offer a story-centric paradigm in contrast to traditional behavior authoring processes which either focus on authoring individual decision processes for each actor, or wholly centralized directors that continuously dispatch commands to every actor in the scene. Stories in our system are naturally formed from sequences of events containing all the behavior necessary for their participants to carry out the plot in simulation.

Second, we have developed a system for allowing virtual characters to smoothly transition from being simulated individually, to performing as members of a singular Smart Crowd entity. Numerous such crowds can exist in the environment, each with their own collective behavior and purpose, and individual characters can join and leave these crowds freely. Our crowds are not simply ambient scene decoration. Rather, our Smart Crowd entities can be controlled in our stories as simply as if they where a single character, and are capable of participating in interactions with other characters or Smart Crowds. They can be controlled to play a significant narrative role in an authored story, or individual members can be removed from the crowd at will to be featured as more principal characters with no loss in fidelity.

Contributions. This paper contributes an authoring tool with broad freedom in designing narratives that take place in sophisticated and populous virtual environments full of actors and objects. Our simulation system enables the control of over 100 entities, all of which are capable of being either featured as principal characters in an authored plot or controlled by one of several centrally-managed crowds using our Smart Crowd system. Events provide an accessible metaphor for untrained actors to design stories based around a desired plot, rather than burdening the story creator with the problem of either designing character behavior (individual or centralized) or converting a story into a set of goals for some solver to achieve. Our entire system operates at interactive rates, and the process of converting an authored story into an on-screen simulation is instantaneous with no compilation delays – making our system ideal for applications in rapid prototyping, pre-visualization, and storyboarding.

2 Related Work

The maturity in the simulation and animation of virtual characters [Kallmann and Kapadia 2016; Kapadia et al. 2015b] has opened the possibilities for authoring complex animations in large crowds. These methods represent different tradeoffs between the ease of user specification and the autonomy of behavior generation.

Simulation-centric authoring. Crowds literature is rich with many proposed solutions that push the boundaries of simulating believable human crowds. These include particle dynamics [Reynolds 1987], social forces [Helbing and Molnár 1995], velocity-based approaches [Paris et al. 2007; Kapadia et al. 2009], and planning-based approaches [Singh et al. 2011a; Singh et al. 2011b; Kapadia et al. 2013]. Commercial software such as Massive and Golaem are simulation-centric where animators author the responses of an autonomous agent to external stimuli and tweak simulation parameters – molding the emergent crowd behavior to conform the required specifications. This mode of authoring requires the animator to work within the limits imposed by simulation framework, which may be sufficient for generic crowds in the background of a shot (e.g., stadiums, large processions etc.). However, authoring precision is particularly important for crowd shots that mandate interactions with principal characters, or when the behavior of a crowd must be carefully choreographed due to its relevance to the plot. As a result, animators resort to manual methods that provide precise control, at the expense of significantly increasing the authoring burden.

Data-centric authoring. The work in [Kim et al. 2009; Kwon et al. 2008; Kim et al. 2014] synthesizes synchronized multi-character motions and crowd animations by performing editing and stitching operations on a library of motion capture data. Using this approach, the user can interactively manipulate the motions of many characters, while having precise control over individual trajectories. However, the approach is limited to the database of pre-recorded clips as large deformations and time warping may yield unnatural results. Also, editing an individual motion may change the entire crowd animation, which is undesirable when orchestrating crowd activities with multiple constraints.

Motion patches [Lee et al. 2006] provide environmental building blocks that are annotated with motion data, which informs what actions are available for animated characters within the block. Motion patches can be edited and connected together [Kim et al. 2012; Li et al. 2012], or precomputed by expanding a search tree of single character motions [Shum et al. 2008] to synthesize complex multi-character interactions. This concept is generalized to crowds [Yersin et al. 2009] that can serve as building blocks for sophisticated crowd authoring tools [Jordao et al. 2014]. Recent work [Kapadia et al. 2016; Kapadia et al. 2015a; Kapadia et al. 2015c] has shown the promise of visual storyboard representations

and computational intelligence to create compelling animated, interactive narratives.

Behavior-centric Authoring. The aforementioned contributions rely on pre-recorded (or pre-computed) motion building blocks that can edited and concatenated to synthesize animation of long duration, with many characters in complex environments. In contrast, behavior-centric approaches use logical constructs [Shoulson et al. 2011; Yu 2007] to represent decision-making in individuals as well as between interacting actors. Automated planning approaches [Shoulson et al. 2013; Kapadia et al. 2011] can then be used to synthesize complex narratives between a small number of principal characters for interactive narrative applications [Riedl and Bulitko 2013]. These approaches rely on autonomy to generate emergent stories while sacrificing authorial precision.

Comparison to Prior Work. Our research aims to generalize motion patches so as to represent logical behavioral constructs that can be stitched together with precise authorial control, while using simulation to automatically synthesize interaction instances that meet author constraints. In particular, we use use Parameterized Behavior Trees (PBTs) [Shoulson et al. 2011] to author modular interactions between multiple characters, and present an authoring platform that allows untrained users to generate complex narratives. Our work uses a similar story representation and visual story-boarding interface to [Kapadia et al. 2016] and extends it to include crowd-level control, facilitating the seamless transition between background and foreground characters for storytelling.

3 Preliminaries

3.1 Terminology

Our framework requires domain knowledge specified by experts in order to use automation for computer-assisted authoring. This includes annotating semantics that characterize the different ways in which objects and characters interact (affordances), and how these affordances are utilized to create interactions of narrative significance (events), which serve as the atoms of a story. Our system is no different from other intelligent systems [Riedl and Bulitko 2013] in this regard. However, the cost of specifying domain knowledge is greatly mitigated by the ability to author a variety of compelling narratives in a fashion that is accessible to story writers, artists, and casual users, enabling authors to focus only on key plot points while relying on automation to facilitate the creative process. We describe our representation of domain knowledge which balances ease of specification and efficiency of automation. Our representation of domain knowledge follows an event-centric representation, similar to [Kapadia et al. 2016].

Smart Objects and Actors. The virtual world $\mathcal{W}$ consists of smart objects with embedded information about how an actor can use the object. We define a smart object $w \in \mathcal{W}$ as $w = \langle \mathbf{F}, s \rangle$ with a set of advertised affordances $f \in \mathbf{F}$ and a state s. Smart actors inherit all the properties of smart objects and can invoke affordances on other smart objects or actors in the world.

Smart Crowds. A Smart Crowd $w_g = \langle \mathbf{F}, s, i\rho, \mathbf{M}, \mathcal{E} \rangle$ is a special kind of Smart Actor that also contains its own Affordances, State, ID and Routine. In addition, a Smart Crowd contains a mutable set of Smart Object Members $\mathbf{M} \subset \mathcal{W}$ ($w_g \notin \mathbf{M}$), and an Event Lexicon $\mathcal{E}$ – a set of authored Events that it uses to control its Smart Object Members $\mathbf{M}$ during Affordance activations $f \in \mathbf{F}$ (or when the Smart Crowd is instructed to activate another Smart Object's advertised Affordance). A Smart Crowd's job is to coordinate and conduct its member objects' behavior by dispatching commands and Events from its lexicon. Like a Smart Actor, a Smart Crowd has its own Routine to maintain the baseline activity of its members. Rather than using PBTs, however, a Smart Crowd Routine's decision process is distribution-based. The Routine continuously dispatches low priority Events chosen from a set of possible Events.

Affordances. An affordance $f(w_o, w_u) \in \mathbf{F}$ is an advertised capability offered by a smart object that takes the owner of that affordance w_o and another smart object user (usually a smart actor) w_u, and manipulates their states. Reflexive affordances $f(w_o, w_o)$ can be invoked by the smart object owner. For example, a chair can advertise a "Sit" affordance that controls an actor to sit on the chair.

State. The state $s = \langle \theta, R \rangle$ of a smart object w comprises a set of attribute mappings θ, and a collection of pairwise relationships R with all other smart objects in $\mathcal{W}$. An attribute $\theta(i, j)$ is a bit that denotes the value of the j^{th} attribute for w_i. Attributes are used to identify immutable properties of a smart object such as its role (e.g., a chair or an actor) which never changes, or dynamic properties (e.g., `IsLocked`, `IsIncapacitated`) which may change during the story. A specific relationship $R_a(\cdot, \cdot)$ is a sparse matrix of $|\mathcal{W}| \times |\mathcal{W}|$, where $R_a(i, j)$ is a bit that denotes the current value of the a^{th} relationship between w_i and w_j. For example, an `IsFriendOf` relationship indicates that w_i is a friend of w_j. Note that relationships may not be symmetric, $R_a(i, j) \neq R_a(j, i)\ \forall\ (i, j) \in |\mathcal{W}| \times |\mathcal{W}|$. Each smart object's state is stored as a bit vector encoding both attributes and relationships. The overall state of the world $\mathcal{W}$ is defined as the compound state $\mathbf{s} = \{s_1, s_2 \cdots s_{|\mathcal{W}|}\}$ of all smart objects $w \in \mathcal{W}$. $\mathbf{s_w}$ denotes the compound state of a set of of smart objects $\mathbf{w} \subseteq \mathcal{W}$. The compound state of all smart objects in the world is encoded as a matrix of bit vectors.

Events. Events are pre-defined context-specific interactions between any number of participating smart objects where each instance of an event can have a vastly different outcome depending on the participating smart objects and their current state. Events serve as the building blocks for authoring complex narratives. An event is formally defined as $e = \langle t, \mathbf{r}, \Phi, \Omega, c \rangle$ where t is a Parameterized Behavior Tree (PBT) [Shoulson et al. 2011] definition, and is an effective model for representing coordinated behaviors between multiple actors. t takes any number of participating smart objects as parameters where $\mathbf{r} = \{r_i\}$ define the desired roles for each participant. r_i is a logical formula specifying the desired value of the immutable attributes $\theta(\cdot, j)$ for w_j to be considered as a valid candidate for that particular role in the event.

A precondition $\Phi : \mathbf{s_w} \leftarrow \{\texttt{TRUE}, \texttt{FALSE}\}$ is a CNF expression on the compound state $\mathbf{s_w}$ of a particular set of smart objects $\mathbf{w} : \{w_1, w_2, \cdots w_{|\mathbf{r}|}\}$ that checks the validity of the states of each smart object. The literals in the CNF expression include desired attributes of each smart object, relationships, as well as rules between pairs of participants. A precondition is fulfilled by $\mathbf{w} \subseteq \mathcal{W}$ if $\Phi_e(\mathbf{w}) = \texttt{TRUE}$. The event postcondition $\Omega : \mathbf{s} \rightarrow \mathbf{s}'$ transforms the current state of all event participants $\mathbf{s}$ to $\mathbf{s}'$ by executing the effects of the event. When an event fails, $\mathbf{s}' = \mathbf{s}$. The event cost c determines the cost of executing an event and can be author-specified or automatically derived based on its postconditions.

While most event parameters are explicitly set by the author or automation system, some can be implicitly inferred from explicit parameters (e.g. a character can only sit on one chair at a time, so involving the chair that character is sitting on in an event can be done implicitly). Functionally, both types of parameters are equal – Φ_e evaluates their state and Ω_e modifies their state. However, implicit parameters are hidden from the author and are only needed for automation.

Event Instances. An event instance $I = \langle e, \mathbf{w} \rangle$ is an event e populated with an ordered list of smart object participants $\mathbf{w}$. I is: (1)

complete iff $r_i(w_j) = \texttt{TRUE}\ \exists\ w_j \in \mathbf{w},\ \forall\ r_i \in \mathbf{r}_i$, and (2) *valid* iff $\Phi_e(\mathbf{w}) = \texttt{TRUE}$.

Beats and Story Arcs. A Beat $\mathbf{B} = \{I_1, \ldots, I_n\}$ is a collection of Event Instances for Events happening simultaneously at a particular point in the story. A Story Arc $\alpha = (\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_m)$ is an ordered sequence of Beats representing a story, where Events can occur both sequentially and simultaneously throughout that story's execution. We refer to the set of all Event Instances across all beats as $\mathbb{I} = \mathbf{B}_1 \cup \mathbf{B}_2 \cup \ldots \cup \mathbf{B}_m$. For the Beat $\mathbf{B}_i$ to start, it must either be the first beat in the Story Arc, or the Beat $\mathbf{B}_{i-1}$ must have already terminated. A Beat is said to have terminated once all its Event Instances have terminated.

3.2 Character Control Hierarchy

To ensure realistic animation, every character within the scene should be kept busy even when not actively involved in a story. Principal characters that contribute to the narrative are simulated individually, while non-focal background characters are grouped together and managed by Smart Crowds that are capable of their own collective autonomy. To control these characters with varying degrees of importance, we designed a priority system with three levels of control where each character can seamlessly transition between levels in the hierarchy. This hierarchy is illustrated in Figure 2.

Individual Routines. As a Smart Actor, each individual character has a default Routine that runs continuously when the character is involved in neither the narrative nor a Smart Crowd. This is the behavior with the lowest priority and simply keeps the character active and moving. Individual character routines are not aware of other characters and are thus incapable of causing interactions between characters at will. Examples of these types of behavior are wandering and looking around.

Crowd Routines and Crowd Events. If a character is part of a Smart Crowd, the Smart Crowd object determines his or her actions by dispatching Events within the crowd. Events dispatched by the Smart Crowd have a higher priority than the default individual Routines of its members, and are capable of dictating more sophisticated behavior such as interactions between characters of the same crowd. Just as a Smart Actor will act autonomously if not involved in the story, a Smart Crowd will execute its own Routine when acting independently – autonomously picking Events from its lexicon and dispatching them to its member Smart Objects. Examples of Smart Crowd interactions are crowd members talking to each other or buying goods from a marketplace.

Story Events. Arc-specific Story Events are the events dictated by the authored Story Arc's Beats. When dispatched, these events have the highest priority and are used to control the actions of the principal characters and relevant crowds in order to further the storyline. They are authored in the Story Arc interface, and every Smart Object (including Smart Crowds) in the world can be controlled with Story Events.

3.3 Priority Conflicts and Event Scheduling.

Story Events can control both crowds and characters directly in service to the plot. However, inter-event conflicts can arise within even a single Story Event. If a Smart Crowd is authored to do something (i.e., if it has one of its Affordances activated, or the Story Event commands the Smart Crowd to activate some other Smart Object's Affordance), it will dispatch Crowd Events to its member Smart Objects to direct them through some activity. However, members of the crowd can also be selected individually to participate in the Story Event directly as principal characters. For example, the Story

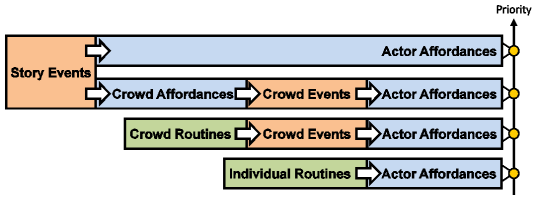

Figure 2: *The priority hierarchy of control structures illustrating how commands are translated from Routines and Events until finally reaching each individual character in the scene in the form of Affordance activations.*

Event could tell the Smart Crowd to have all of its members turn and watch a central stage area, while the Story Event also tells two members of that same Smart Crowd to approach the stage and begin performing. This creates a conflict for these characters between the Crowd Events dispatched by their controller Smart Crowd, and the Story Event that also wants to control them directly. In order to solve this conflict, the events that are dispatched by the Smart Crowd have a lower priority. This means Smart Crowds can be controlled by Story Events, but members of that Smart Crowd can be taken away at any point to participate in a higher priority Story Event. As a result, minor narrative elements of the crowd do not conflict with the performance of principal characters even if they were previously members of that Smart Crowd.

Since Events can pre-empt one another according to priority, it is possible for one Event to abruptly terminate another lower priority Event when it needs some or all of that low-priority Event's participants. As such, we construct our Events in a way so that they may be terminated at any time. When a higher-priority Event is scheduled for a Smart Object that is already part of a lower priority Event (or is acting according to its own autonomous Routine), the lower priority Event is terminated in a clean fashion before the new Event can start. This is handled by the PBT structure of the Event itself, which can send special termination signals to clean up the current set of actions and reach a safe termination state. One special case is with Smart Crowd Events, which are explained in more detail in Section 4. With Crowd Events, if a participating Smart Object receives a new Event, the Crowd Event can relinquish control over that single Smart Object instead of terminating entirely. Those Smart Objects are then excluded from any subsequent lower-priority Events until their high-priority Event terminates (despite still being members of that Smart Crowd). This ensures we do not have to terminate a Crowd Event with a high number of participants due to a single or few participating Smart Objects being promoted to a higher priority Event.

3.4 Expertise of Authoring Tasks

Creating an entirely new story setting is split into three main task families, each requiring their own level of expertise.

World Engineering. At the lowest level, a new scenario requires the programming and scripting of each Smart Object and its Affordances. This currently has no authoring tool and still requires the direct writing of code, as well as the management of model, texture, and animation assets for the objects and the world environment itself. We expect the world engineer to be an expert author with programming experience.

Behavior Authoring. The middle level of the scenario revolves around the construction of PBTs. A trained, but not expert user should be capable of authoring PBTs for individual character Routines, as well as designing both Crowd and Story events using Smart

Object Affordances. PBT authoring is currently also done in code, but with a greatly simplified and more accessible syntax and structure compared to open-ended software development.

Story Design. The top level of the scenario accessible to end-users is the story authoring itself. The story author places actors, crowds, and objects in the world and then uses a simple drag-and-drop authoring GUI to create Story Arcs using these world elements. We have designed this aspect of the story specification to be as accessible as possible. It is also possible that an advanced story designer would be able to author some Story Events or other PBTs directly after gaining experience with the system. The story author can also set the distribution and decision process for the Routines used by Smart Crowds to dispatch Crowd Events autonomously.

4 Smart Crowds

Recall that we define a Smart Crowd $w_g = \langle \mathbf{F}, s, i, \rho, \mathbf{M}, \mathcal{E} \rangle$.

Affordances $\mathbf{F}$: Like any Smart Object, a Smart Crowd advertises Affordances to be used by other objects. These represent capabilities of the crowd collectively as a whole. When invoked, Smart Crowd Affordances can dispatch Crowd Events that invoke the Affordances of the Smart Crowd's Members $\mathbf{M}$. Similarly, when a Smart Crowd is told to activate another Smart Object's Affordance, it dispatches Crowd Events instructing all of its members to do the same. For example, when the Smart Crowd is instructed to activate another Smart Object's "LookAt" Affordance, it dispatches a Crowd Event to all of its members to look at the target object (i.e. each member of the Smart Crowd uses the target Smart Object's "LookAt" Affordance individually during the Crowd Event).

State s and **ID** i: Just as any other Smart Object, a Smart Crowd also contains a State and has its own unique ID. The Crowd's State refers to itself, and does not consider the State of any of its Members $\mathbf{M}$.

Routine ρ: When acting autonomously, the Smart Crowd schedules its own internal Crowd Events using an authored distribution of desired activity. In this situation, the goal of the Smart Crowd is to maintain a baseline of interesting actions and interactions for its members to perform. The Smart Crowd's Routine is discussed in more detail in Section 4.2.

Smart Object Members $\mathbf{M}$: A Smart Crowd contains many Smart Objects of varying types (a Smart Crowd can contain, for example, both a number of actors and a number of tables and chairs). A Smart Crowd can not contain other Smart Crowds. Thanks to the hierarchy of Smart Objects, Affordances, and Events, designing an interface for Smart Crowds that encapsulate groups of other Smart Objects is rather straightforward – Events can activate Affordances on Smart Crowds (or command Smart Crowds to activate other Affordances), and in turn the Smart Crowd dispatches Events that activate Affordances on the crowd's members.

Event Lexicon $\mathcal{E}$: Each Smart Crowd has an authored set of Events with which it can control its members. These Events are linked to Affordances, and will be dispatched when certain Affordances are activated (either by or on the Smart Crowd). Crowd Events are flexible so as to accommodate the different types of Smart Objects that can be can be contained in a particular Smart Crowd, and the Smart Crowd will automatically filter through the Roles of its available members (i.e. those not currently involved in another Event of equal or higher priority) for valid participant candidates when picking a Crowd Event to dispatch. As a Smart Object contained in a Smart Crowd could be promoted to a higher priority event at any given time, there are certain limitations on the Events included within the Event Lexicon. These Events may not change the state of its participating Smart Objects in any way, so that any Smart Object temporarily dropped off into a Crowd can be used again later without having its state changed in the meantime. Note however, that such an Event may change state in an intermediate step, however when being terminated it must make sure to undo any changes to the state of the participating Smart Objects to be considered for the Event Lexicon. Examples of Crowd Events include conversations between two (or more) characters, characters temporarily sitting on benches and chairs, and characters haggling with a vendor to purchase items from a stall.

4.1 Extending PBTs for Crowds

Our work with Smart Crowds exposed a limitation of the PBT formalism that made it difficult to manage large malleable groups of characters. A key quality of traditional PBTs is that they require a fixed set of individually specified participants in order to function. As a result, a conversation designed for two actors (for example) could not be trivially reused with three, or with two actors and a bench. Additionally, traditional PBT Events co-opt the control of their participants for the entire duration of the Event, even if one participant is no longer needed. If a higher priority Event requests control of a character currently involved in another low-priority traditional Event, the low-priority Event must be terminated entirely to release that character even if the character is not essential to the low-priority Event's execution. This was not amenable to creating large Events controlling potentially dozens of characters at a time, where characters might enter and leave the Smart Crowd at arbitrary points.

To enable this new kind of functionality, we extended our PBT system with the ForEach node. The ForEach node handles Event behavior for a dynamically changing set of Smart Objects. A ForEach node starts with a given set of Smart Objects which it may control. Over the course of the Event, the ForEach node can release any of these Smart Objects without terminating the whole node itself. With this new type of node, we can cleanly resolve issues occurring when a character must leave the Smart Crowd in the midst of an Event because it has been promoted to a higher priority Event. For example, a Smart Crowd can perform some macroscopic activity while one of its members is selected individually to leave and play a principal role in some Story Event. This character extraction does not interfere with the execution of the Crowd Event itself (unless that removed member was the only remaining member capable of performing the Crowd Event overall). The following algorithms illustrate the initial setup of a ForEach node as well as a single tick of the node. Note that the algorithm for a single tick of the ForEach node is slightly simplified here, as terminating a node can take more than one tick.

Input: Initial object list $\mathbf{I}$
Input: Function to create nodes $\mathbf{F}$
Result: Mapping of objects to nodes $\mathbf{M}$
foreach $i \in \mathbf{I}$ **do**
    $\mathbf{M}[i] = \mathbf{F}(i)$;
end

Algorithm 1: *Initialization of a ForEach node.*

Input: New object list $\mathbf{N} \subseteq \mathbf{C}$
Data: Current object list $\mathbf{C}$
Data: Mapping of objects to nodes $\mathbf{M}$
Result: Current object list for next tick $\mathbf{C}$'
foreach $c \in \mathbf{C} \setminus \mathbf{N}$ **do**
    terminate $\mathbf{M}$ [c] ;
end
foreach $n \in \mathbf{N}$ **do**
    tick $\mathbf{M}$ [n] ;
end
$\mathbf{C}$' = $\mathbf{N}$

Algorithm 2: *A single tick of a ForEach node.*

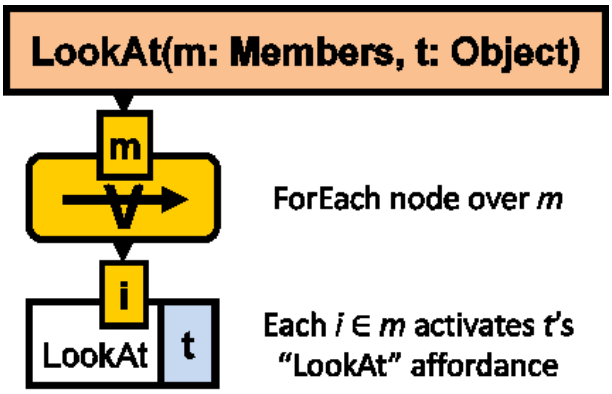

Figure 3: *A simple Crowd Event that uses a ForEach node.*

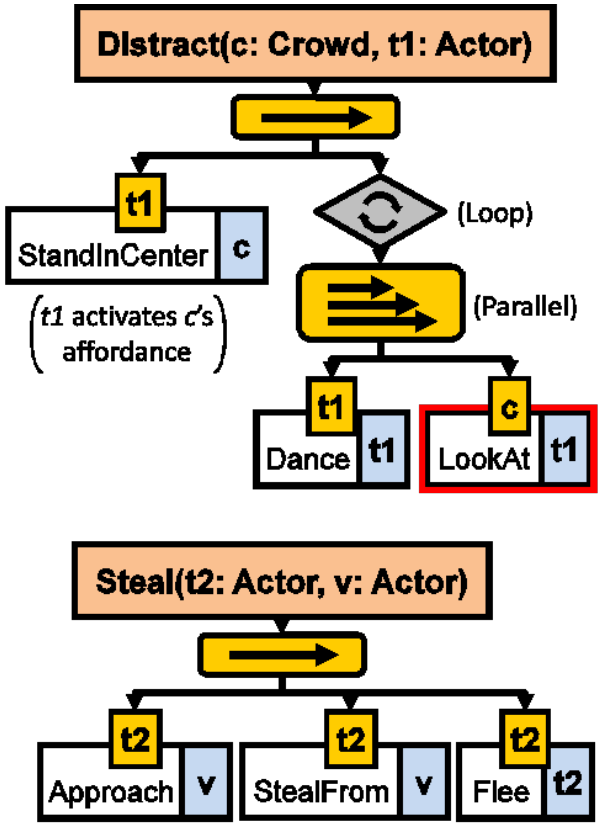

Figure 4: *Two Story Events demonstrating interactions between individuals and crowds.*

Figure 3 illustrates a simple use of the ForEach node in a Crowd Event. Recall that just as a chair advertises a "Sit" Affordance that instructs a user to sit on the chair, most Smart Objects advertise a "LookAt" Affordance that instructs a user to look at that object. When a Smart Crowd activates another object's "LookAt" Affordance, however, it internally dispatches this Crowd Event that gathers all of the Smart Crowd's valid members m (a special parameter type) and instructs them to individually activate that target object t's "LookAt" Affordance.

Figure 4 shows an example of a more complicated use of the ForEach node. In this case we have a Smart Crowd of bystanders c, a victim v, and two thieves $t1$ and $t2$. These two events, when executed sequentially in the story, will create a scenario where $t1$ distracts the crowd while $t2$ steals from the victim v. Note, however, that v is initially part of the Smart Crowd c. In the first event, $t1$ activates an Affordance on the Smart Crowd to stand in the center of the group (this does not dispatch a Crowd Event as the Smart Crowd itself takes no action – it merely instructs $t1$ to approach its center point). Then $t1$ begins dancing, and the Smart Crowd is instructed to look at $t1$ by activating $t1$'s "LookAt" Affordance (node highlighted in red). This Affordance activation causes c to dispatch a Crowd Event to its members (the same one illustrated in Figure 3). Importantly, since v is a member of c, v is included in the Crowd Event's ForEach node, and also looks at $t1$ with the rest of the crowd. Afterwards, while *Distract* is still running, we activate the *Steal* event, which uses v as an individual character. In this case, v is seamlessly removed from the ForEach node, and used in the *Steal* event, where $t1$ approaches v, steals an item (by activating v's "StealFrom" Affordance), and reflexively activates its own "Flee" Affordance to escape. In traditional PBT events, v would have been locked in the *Distract* Crowd Event, and could not have been extracted for use in the *Steal* event without terminating *Distract* (causing $t1$ to stop dancing, and the crowd to potentially stop watching).

4.2 Defining Smart Crowds and their Activity

The members of a crowd are selected by selecting a spatial region in the world. All Smart Objects contained within that spatial region (apart from other Smart Crowds) will then be part of the crowd. It is possible to make the crowd's members static or not. If the members are static, then the Smart Crowd's members are evaluated only once and will stay a part of that crowd for the whole narrative. If the crowd's members are not statically determined, characters can both enter and leave the crowd at any time by entering or exiting the Smart Crowd's defined spatial region. An individual character may wander into a Smart Crowd's zone while running its individual Routine, at which point it will suspend its own Routine and begin receiving Crowd Events from the Smart Crowd (unless involved in a Story Event). Similarly, members leaving the zone will restart their individual Routine and act autonomously with no Crowd Events. It is also possible for these spatial regions to overlap, so any Smart Object can potentially be a member of arbitrarily many Smart Crowds. All the Smart Crowds that contain the member can then try and use that Smart Object for Crowd Events.

When running its own Routine, a Smart Crowd schedules events among its members that satisfy an author-specified distribution based on Group Coordination [Shoulson and Badler 2011]. The lexicon of events and its distribution may differ within a crowd depending on the types of (Smart) actors and objects it encompasses. For example, one Smart Crowd might have benches, allowing the characters to sit down, while another one does not, and so Events

requiring the "Bench" role would be invalid for the latter crowd. The Smart Crowd's Event scheduler Routine uses an authored distribution of a given set of events, and tries to find available matches from within the crowd's objects while also making sure not to interrupt any objects participating in equal or higher priority Events. To dispatch a new Event, the Smart Crowd first selects an authored event from its Lexicon, and then greedily populates the Event's participant list with its members, filtering according to the following two criteria: (1) Do the objects have the correct role and state, and (2) are all the objects currently available to participate in an event of the given priority. If both of these criteria are satisfied, then an event instance is dispatched with the selected participants.

5 User Interface

Story Arcs are authored using Story Sequence Diagrams specified in our graphical authoring tool. A Story Sequence Diagram is a directed acyclic graph $Q = \langle V, E \rangle$. The vertices $V = V_S \cup V_I$ consist of Smart Objects $V_S \approx \{w_1, \ldots, w_n\} \subseteq \mathcal{W}$, and Event Instances $V_I \approx \{I_1, \ldots, I_m\} \subseteq \mathbb{I}$. The edges $E = E_\pi \cup E_\sigma \cup E_\varphi$ indicate three relationship types. Participation Edges $E_\pi \subseteq V_S \times V_I$ denote a "participates in" relationship between a Smart Object and an Event Instance (to populate a Role in the Event Instance that is satisfied by the Smart Object). Sequence Edges $E_\sigma \subseteq V_I \times V_I$ denote a "comes after" relationship between two Event Instances that specify temporal restrictions. Termination Edges $E_\varphi \subseteq V_I \times V_I$ denote a termination dependency, where $(I_i, I_j) \in V_I \times V_I$ denotes that I_j is terminated as soon as I_i has terminated. Note that I_i and I_j must be in the same Story Beat, as otherwise they will never be running at the same time. Sequence Edges can be manually added by the author to define separate story Beats, or are automatically inserted into the Story Sequence Diagram when a Smart Object participating in the Event Instance is already involved in another Event at the same time. Each horizontal row of Event Instances delineates a Beat (two or more Event Instances that can occur simultaneously), and the ordered sequence of Beats represents the resulting Story Arc. A diagram of a simple Story Arc represented in the GUI can be seen in Figure 5.

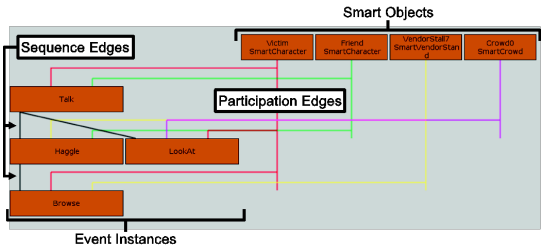

Figure 5: *A simple Story Arc displayed in the GUI showing different edge and vertex types. Note that Termination Edges are not explicitly displayed.*

Story Arc Authoring. Once the author has added Smart Objects, Actors, and Crowds to the environment, authoring a Story Arc from a library of Events requires a few simple steps.

1. Add as many Story Events as needed by dragging them into the main window from the sidebar. Story Events can be added to the window at any time.
2. Add as many Smart Objects as needed by dragging them into the main window from the sidebar. Smart Objects can be added to the window at any time.
3. Drag the Story Events into the position within the Story Arc where they should be executed.
4. Connect the participating Smart Objects to their respective Story Events by clicking and dragging to create connection lines.
5. Optionally, instead of manually creating the connections from the Smart Objects to the Story Events, the automation capabilities can be used to automatically fill in any gaps left by the author.

Once a Story Arc has been completed, it can be executed immediately in simulation. There is no expensive compilation, serialization, or build step.

Validity Constraints. The Story Arc interface enforces a number of restrictions on Story Event sequences and their participants in order to guarantee the arc's validity and prevent errors. First, the interface ensures that the Smart Object participants of an Event are valid, meaning their Roles correspond to the required Roles of the Event. Invalid Smart Objects can not be added to an Event. Secondly, the interface ensures that a Smart Object can not participate in two Events within the same Story Beat, unless it is specifically marked as a Non-Participant in one of the Events, meaning that its role within that Event is entirely passive (for example "being looked at"). If a Smart Object is linked to multiple Events in the same Beat, one of those events is automatically moved to the next Beat. While the GUI itself only makes sure the Smart Objects satisfy the necessary roles, but not any dynamic state or relationships, the integration of planning and validity checking in the GUI allows us to also check whether Smart Objects satisfy their dynamic state and relationship preconditions, as well as injecting new Story Events into the Story Arc where necessary.

Smart Crowd Selection. The GUI also offers a way of selecting Smart Crowds using a spatial area selection. Crowd selection is done in two steps. In a first step, a spatial zone within the world space is selected. Also, the author can select whether the Crowd he is authoring should select its members on a static basis or not. In a second step, the Smart Crowd's Event Lexicon is configured. As mentioned before, the Event Lexicon consists of a selection of Events and a probability distribution assigning a probability to each Event within the Lexicon to be selected by the Crowd's event scheduler, as well as a fixed time step to indicate the frequency of Crowd Events being dispatched. The author can manually select the Events to be considered in the Lexicon and assign a probability to each of them, as well as select the update time step. The Events listed in this window are a subset of all available Events consisting of exactly those Events that do not change the state or relations of any of their participants, i.e. do not have a postcondition.

6 Authored Narrative

6.1 World

We used the following Smart Objects in our example narratives:

Smart Chair. A chair or a bench where characters can sit down. Smart Chairs have the "chair" Role and the following Affordances: (1) Sit: The user (an Actor) is directed to sit on the chair. (2) Stand: The user stands up from the chair.

Smart Vendor Stand. A vendor stand with fruit and a vendor that sells them. Smart Vendor Stands have the "VendorStand" Role and the following Affordance: (1) Haggle: The user approaches the stand and haggles with the vendor. This Affordance is special in that it takes an argument that determines whether or not the vendor successfully sells an item to the user.

Smart Character (i.e. Smart Actor). Each mobile character of the scene is a Smart Character. Smart Characters have the "Actor" Role, and some individuals have more specific roles like "Thief"

and "Hero" (with their own specialized Affordances). The generic Smart Character has the following Affordances (as well as a few not listed here):

- Give: The user gives this character an object.
- Steal: The user approaches this character and steals an object.
- CallOut: The user calls out to this character to get its attention.
- TalkHappily: The user talks happily to this character.
- TalkSecretive: A conversation Affordance that uses more subtle gestures so as to be less noticeable.
- Approach: The user approaches this character.
- Chase: The user runs towards this character.
- Depart: The user walks away from this character.
- LookAt: The user looks at this character.
- BackAwayFrom: The user backs away from this character if close enough.
- LookAround: A reflexive Affordance where the character looks around the scene.
- Eat: A reflexive Affordance that lets the character eat what it is holding.

Smart Crowd. Smart Crowd objects have the "Crowd" Role and represent a special case of Affordance activations compared to single objects. When a Smart Crowd activates an Affordance on a single object (as opposed to another Smart Crowd), it internally dispatches a Crowd Event to its members instructing them to all individually activate that same Affordance on the target object (see Figure 3 for example of one of these Events. This trivially enables some group behaviors such as "Gather Around" (where the crowd's members all activate an object's "Approach" Affordance), or "React To" (where they activate an object's "BackAwayFrom" Affordance).

6.2 Story Events

We used the following Story Events in our Story Arcs (among others):

Browse(Actor, VendorStand): Instructs an actor to inspect a vendor stand, haggle with the vendor, purchase an apple, and eat it while looking around.

DropAndReturn(Actor, Actor): The first actor drops its wallet while walking. The second actor approaches the first and returns the wallet. Figure 6 illustrates this Event in more detail.

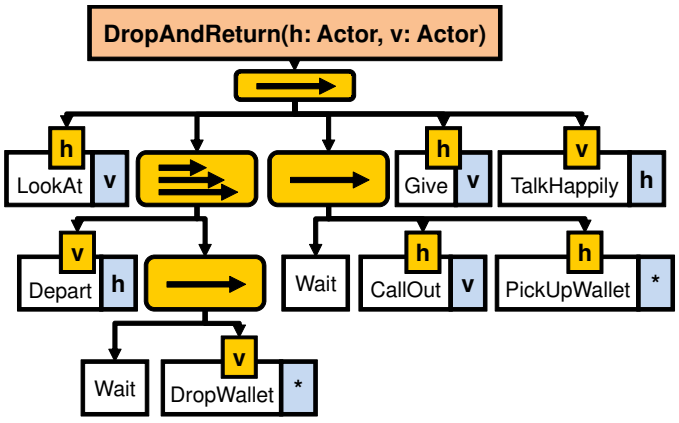

Figure 6: *The PBT of the DropAndReturn Event with a hero h and victim v. Note that DropWallet and PickUpWallet are special Affordances that manage a Wallet prop (which is not a SmartObject).*

Converse(Actor, Actor, ...*): Two actors converse with each other. There are many variations of a default conversation (**ConverseWhileSitting**, **ConverseSecretive**, etc.). Certain conversations can take more actors or more specific actor types.

CallAndConverse (Actor, Actor): The first actor sees the second and calls out, whereupon the second actor approaches the first and they engage in a conversation. Figure 7 illustrates this Event in more detail.

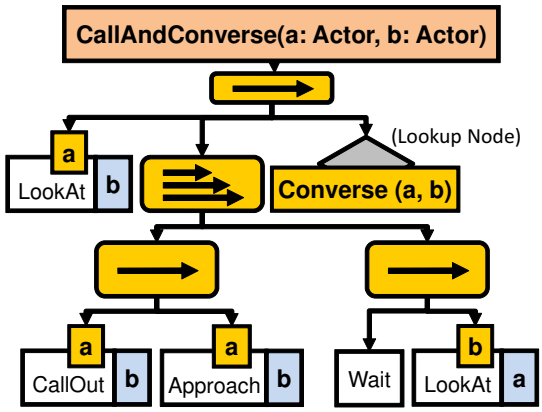

Figure 7: *The PBT of the CallAndConverse Event with two characters a and b. This uses a lookup node to insert a reusable sub-tree for the actual conversation gestures.*

GetAttention(Actor): The actor shouts and tries to get the attention of the people around it.

Gather(Crowd, Any): The crowd looks at and gathers around a particular Smart Object (using that object's Approach Affordance).

Perform(Actor): The actor starts to break-dance for a certain time.

LookAt(Actor, Any): The Actor looks at the object of interest.

Steal(Thief, Actor): The thief approaches target the from behind and steals an item.

React(Crowd, Actor): The crowd looks at the target and backs away if too close.

Chase(Actor, Actor, Thief): The thief runs away while the first actor chases. The second actor approaches from another angle and the two actors corner the thief. The thief surrenders. Figure 8 illustrates this Event in more detail.

6.3 Crowds

Our example scene had four distinct Smart Crowds, each defined according to control zones as illustrated in Figure 9. Actors entering or leaving these regions would automatically join or leave their respective crowds. In addition, each crowd contained static objects like vendor stands and benches that it could use in its Events. All four crowds were of the same type with the same Routine for default behavior. Our crowds' Routines automatically dispatched the following types of activities for ambient behavior:

- Nearby characters meet and talk. They sometimes wander together while talking.
- Wander around within the region of the crowd.
- Sit down on bench, look around.
- Look around while standing still.
- Go to a vendor stand and haggle with a vendor.

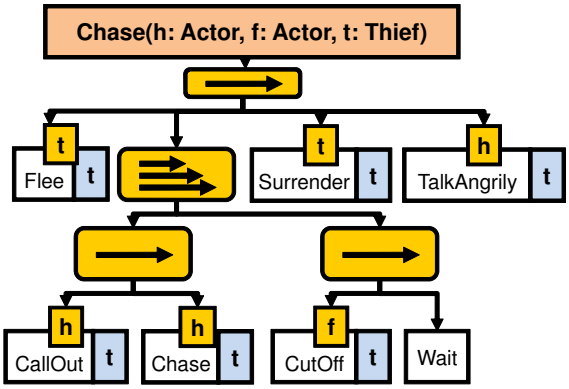

Figure 8: *The PBT of the Chase Event with a hero h, friend f, and thief t. Note that CutOff is a special Affordance that acquires a position for the character to stand to block the escaping thief.*

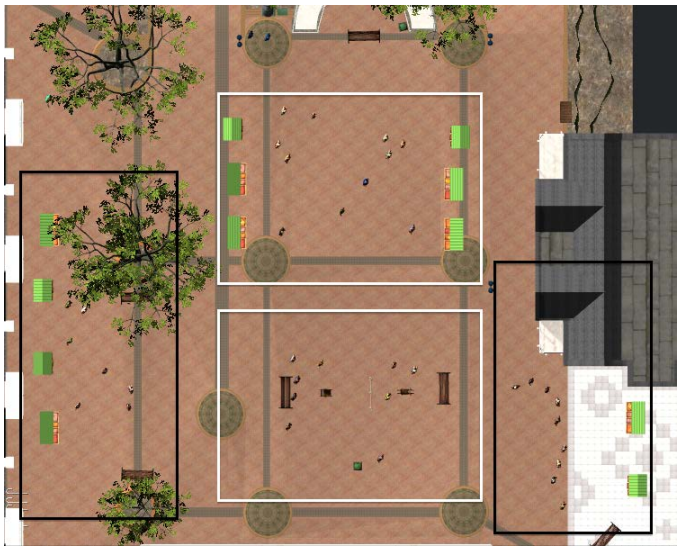

Figure 9: *The spatial regions determining the smart crowds*

- If character is sitting, either look around or talk to character sitting nearby.

6.4 Story Arcs

Story Arc 1: Thief gets away. In the first story arc, the businessman loses his wallet without any other characters noticing. He later finds it again, before it is stolen from a thief. However, no other characters notice the theft. For each story beat, we include the relevant in-engine event names in parenthesis.

- The hero walks into the scene, walks around a bit in the plaza and haggles with a vendor (**Browse**).
- A businessman is seen walking around and dropping his wallet, without anybody noticing (**Drop**). As soon as he notices it to be missing, he starts looking for it (**Search**).
- A friend of the hero spots him (the hero) and calls him over (**CallAndConverse**, see Figure 7). They converse for a bit before deciding to go to a bench to have a more quiet place to talk (**Sit**, **SitAndConverse**).
- After their conversation, two suspicious men secretively converse without anyone noticing them (**TalkSecretive**).
- One of the suspicious men walks to the center of the plaza and gets everyone's attention. Soon, many people have gathered around him, including the businessman (**GetAttention**, **Gather**).
- While the first suspicious man starts break-dancing in order to keep everyone's attention, the second one goes over to the businessman and steals his wallet. (**Perform**, **LookAt**, **Steal**).
- Nobody notices the theft, and once the performance is over, life in the plaza goes back to normal.

Story Arc 2: Thief gets caught. In the second story arc, we will introduce the hero to the victim, so he will be aware of the victim carrying his wallet around and notice the theft. This significant change in story only required minor changes to the event beats in our authoring tool, as follows:

- The story begins as it did in Story Arc 1, until the businessman drops his wallet.
- The hero sees the businessman dropping his wallet. He immediately returns it and the businessman shows his gratitude (**DropAndReturn**, see Figure 6).
- The hero's friend approaches him and the two take their conversation to a park bench as before (**CallAndConverse**, **Sit**, **SitAndConverse**).
- This time, as the two suspicious characters talk secretively (**ConverseSecretively**), the hero watches them. Due to the secretive manner of their conversation, he grows suspicious of them (**LookAt**).
- The two thieves once again gather the crowd's attention and steal from the businessman (**GetAttention**, **GatherAroundPerformer**, **Perform**, **LookAt**, **Steal**).
- As the thief steals the victim's wallet, the hero notices it. He immediately signals his friend, and they both stand up and start chasing the thief (**Chase**, see Figure 8). The people in the plaza are shocked and do their best to stay out of the way of the chase. Eventually, the hero and his friend corner the thief and he surrenders (**React**).

7 Conclusions and Future Work

We have developed a behavior authoring framework and graphical tool that allows untrained designers to easily create complex narrative sequences incorporating both principal individual characters as well as the macroscopic behavior of crowds. Our system provides a simple and manageable semi-centralized control structure that allows large groups of ambient background characters to play active participatory roles in the story as if they were a single character with simple interface commands. As such, environments with any amount of virtual characters are accessible for use in interesting narratives. The use of default Routines on both individuals and crowds level allows our characters to perform meaningful activities without any top-level authoring, reducing the burden for a story designer to constantly ensure baseline activity in the scene. Crowds and characters carry out reasonable sets of actions on their own and, due to our priority resolution system, seamlessly transition to more principal components of a story when recruited for specific events.

Integral to our system is our graphical interface for authoring stories, which allows drag-and-drop authoring of events by drawing the appropriate connections between events and their participants, and gives feedback on the order in which events will be executed. Our authoring system is a powerful tool, but it does have a few limitations:

- The story authoring interface is currently unaware of state and prerequisites, though they do exist in our system. For example,

the "SitAndConverse" Event requires both participants to be sitting in proximity to one another, and will fail if this is not the case. This is checked during simulation, and implied to the author by the Event names, but the tool does not explicitly convey requirements like this to the story author yet.

- Event construction itself is not integrated into our story authoring tool, though it could be certainly be done. Numerous graphical interfaces for editing behavior trees exist, and could be modified into a system for authoring Story and Crowd Events. We have already begun work on such a system.
- Position selection for events continues to be a frustration for our work. Smart Objects implicitly encode relative positions for their Affordances (e.g. where a character should stand to converse with me), but open ended Events like our Chase required some manually authored positions (such as where the friend character should run to cut off the thief as he flees). Some of our stories also required manually authored "GoTo" commands to ensure that Events happened in the areas we intended for them to occur. Our story interface prompts the story author to select positions from the world to serve as parameters to these kinds of Events, but we aim to investigate more comprehensive solutions.

In addition to addressing these limitations, our immediate future work revolves around deeper integration of state, preconditions, and effects into the authoring process. This system introduces a tool for authoring stories entirely by hand, but sometimes requires the author to insert mundane Events such as instructing characters to sit or look at a particular point or object. We aim to introduce a system that makes authoring a semi-automated process, reducing the amount of labor involved and enabling an author to focus solely on Events of narrative significance while offloading the burden of Event selection and parameter population for less important animation sequences.

References

HELBING, D., AND MOLNÁR, P. 1995. Social force model for pedestrian dynamics. *Phys. Rev. E 51*, 5 (May), 4282–4286.

JORDAO, K., PETTRÉ, J., CHRISTIE, M., AND CANI, M.-P. 2014. Crowd Sculpting: A space-time sculpting method for populating virtual environments. *Computer Graphics Forum 33*, 2 (Apr.).

KALLMANN, M., AND KAPADIA, M. 2016. Geometric and discrete path planning for interactive virtual worlds. *Synthesis Lectures on Visual Computing 8*, 1, 1–201.

KAPADIA, M., SINGH, S., HEWLETT, W., AND FALOUTSOS, P. 2009. Egocentric affordance fields in pedestrian steering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, I3D '09, 215–223.

KAPADIA, M., SINGH, S., REINMAN, G., AND FALOUTSOS, P. 2011. A behavior-authoring framework for multiactor simulations. *Computer Graphics and Applications, IEEE 31*, 6, 45 –55.

KAPADIA, M., BEACCO, A., GARCIA, F., REDDY, V., PELECHANO, N., AND BADLER, N. I. 2013. Multi-domain real-time planning in dynamic environments. In *ACM SIGGRAPH/EG SCA*, ACM, 115–124.

KAPADIA, M., FALK, J., ZÜND, F., MARTI, M., SUMNER, R. W., AND GROSS, M. 2015. Computer-assisted authoring of interactive narratives. In *ACM SIGGRAPH I3D*, ACM, 85–92.

KAPADIA, M., PELECHANO, N., ALLBECK, J., AND BADLER, N. 2015. Virtual crowds: Steps toward behavioral realism. *Synthesis Lectures on Visual Computing 7*, 4, 1–270.

KAPADIA, M., ZUND, F., FALK, J., MARTI, M., SUMNER, R. W., AND GROSS, M. 2015. Evaluating the authoring complexity of interactive narratives with interactive behaviour trees. In *Foundations of Digital Games*, FDG'15.

KAPADIA, M., FREY, S., SHOULSON, A., SUMNER, R. W., AND GROSS, M. 2016. CANVAS: Computer-assisted Narrative Animation Synthesis. In *ACM SIGGRAPH/EG SCA*, Eurographics, SCA '16.

KIM, M., HYUN, K., KIM, J., AND LEE, J. 2009. Synchronized multi-character motion editing. In *ACM SIGGRAPH*.

KIM, M., HWANG, Y., HYUN, K., AND LEE, J. 2012. Tiling motion patches. In *ACM SIGGRAPH / Eurographics SCA*, 117–126.

KIM, J., SEOL, Y., KWON, T., AND LEE, J. 2014. Interactive manipulation of large-scale crowd animation. *ACM Transactions on Graphics (SIGGRAPH 2014, To Appear) 33*.

KWON, T., LEE, K. H., LEE, J., AND TAKAHASHI, S. 2008. Group motion editing. In *ACM SIGGRAPH 2008 papers*, ACM, New York, NY, USA, SIGGRAPH '08, 80:1–80:8.

LEE, K. H., CHOI, M. G., AND LEE, J. 2006. Motion patches: building blocks for virtual environments annotated with motion data. In *ACM SIGGRAPH*, 898–906.

LI, Y., CHRISTIE, M., SIRET, O., KULPA, R., AND PETTRÉ, J. 2012. Cloning crowd motions. In *ACM SIGGRAPH/EG SCA*, EG, SCA '12, 201–210.

PARIS, S., PETTR, J., AND DONIKIAN, S. 2007. Pedestrian reactive navigation for crowd simulation: a predictive approach. *Comput. Graph. Forum 26*, 3, 665–674.

REYNOLDS, C. W. 1987. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH*, 25–34.

RIEDL, M. O., AND BULITKO, V. 2013. Interactive narrative: An intelligent systems approach. *AI Magazine 34*, 1, 67–77.

SHOULSON, A., AND BADLER, N. I. 2011. Event-centric control for background agents. In *ICIDS*, 193–198.

SHOULSON, A., GARCIA, F. M., JONES, M., MEAD, R., AND BADLER, N. I. 2011. Parameterizing behavior trees. In *Motion in Games*, 144–155.

SHOULSON, A., GILBERT, M. L., KAPADIA, M., AND BADLER, N. I. 2013. An event-centric planning approach for dynamic real-time narrative. In *Proceedings of Motion on Games*, ACM, New York, NY, USA, MIG '13, 99:121–99:130.

SHUM, H. P. H., KOMURA, T., SHIRAISHI, M., AND YAMAZAKI, S. 2008. Interaction patches for multi-character animation. In *ACM SIGGRAPH Asia*, 114:1–114:8.

SINGH, S., KAPADIA, M., HEWLETT, B., REINMAN, G., AND FALOUTSOS, P. 2011. A modular framework for adaptive agent-based steering. In *ACM SIGGRAPH I3D*, 141–150 PAGE@9.

SINGH, S., KAPADIA, M., REINMAN, G., AND FALOUTSOS, P. 2011. Footstep navigation for dynamic crowds. *Computer Animation and Virtual Worlds 22*, 2-3, 151–158.

YERSIN, B., MAÏM, J., PETTRÉ, J., AND THALMANN, D. 2009. Crowd patches: populating large-scale virtual environments for real-time applications. In *ACM SIGGRAPH I3D*, 207–214.

YU, Q. 2007. *A Decision Network Framework for the Behavioral Animation of Virtual Humans*. PhD thesis, Toronto, Ont., Canada, Canada. AAINR27686.